WE CLAIM:

Sub A2

1. A trace having a multiple entry, single exit architecture.

2. The trace of claim 1, wherein the trace is a complex trace having multiple independent prefixes and a common, shared suffix.

3. The trace of claim 1, wherein the trace is indexed by an address of a terminal instruction therein.

4. A front-end system for a processor, comprising:

an instruction cache system,

an extended block cache system, comprising:

a fill unit provided in communication with the instruction cache system,

a block cache, and

a selector coupled to the output of the instruction cache system and to an output of the block cache.

5. The front-end system of claim 4, wherein the extended block cache system further comprises a block predictor provided in communication with the fill unit and the block cache.

6. The front-end system of claim 4, wherein the block cache is to store traces having a multiple-entry, single exit architecture.

7. The front-end system of claim 4, wherein the block cache is to store complex traces having multiple independent prefixes and a common suffix.

8. The front-end system of claim 7, wherein the extended block cache system further comprises a block predictor to store masks associated with the complex traces, the masks distinguishing the prefixes from each other.

9. A method of managing extended blocks, comprising:

predicting an address of a terminal instruction of an extended block to be used,

Cont A2

determining whether the predicted address matches an address of a terminal instruction of a previously created extended block, and

selecting one of the extended block in the event of a match.

10. The method of claim 9, further comprising creating a new extended block when there is no match.

11. The method of claim 10, wherein the creating comprises:

receiving new instructions until a terminal condition occurs,

assembling the new instructions into an extended block,

determining whether an address of a terminal instruction in the new block matches an address of a terminal instruction of a pre-existing block, and

unless a match occurs, storing the new block in a memory.

12. The method of claim 11, wherein the storing comprises, when an older block causes a match, storing the new block over the old block in a memory if the old block is subsumed within the new block.

13. The method of claim 11, wherein the storing comprises, when an older block causes a match, dropping the new block if the new block is subsumed within the older block.

14. The method of claim 11, wherein the storing comprises, when an older block causes a match, creating a complex block if the new block and the older block share a common suffix but include different prefixes.

15. The method of claim 9, further comprising outputting instructions of the selected block for execution.

16. A processing engine, comprising:

a front end stage to store multiple-entry, single exit traces, and

an execution unit in communication with the front end stage.

17. The processing engine of claim 16, wherein the front-end stage comprises:

an instruction cache system,

an extended block cache system, comprising:

Cont A2

a fill unit provided in communication with the instruction cache system,

a block cache, and

a selector coupled to the output of the instruction cache system and to an output of the block cache.

18. The processing engine of claim 17, wherein the block cache is to store the multiple-entry, single exit traces.

19. The processing engine of claim 17, wherein the extended block cache system further comprises a block predictor provided in communication with the fill unit and the block cache.

Add A2

Add C1